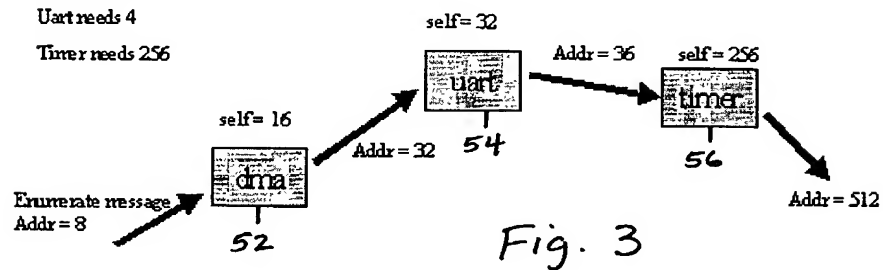
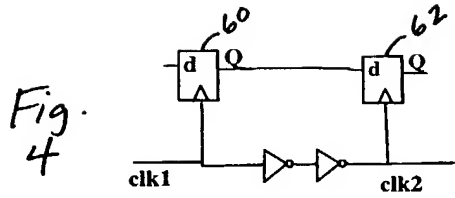


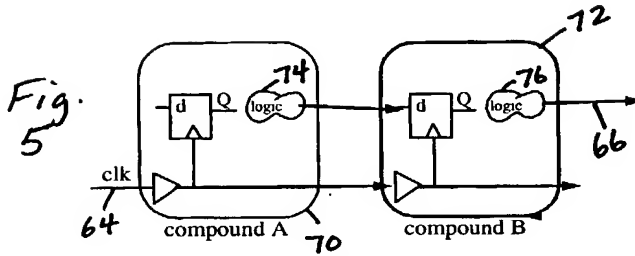
Step4: send downnext\_addr

Time r needs 256



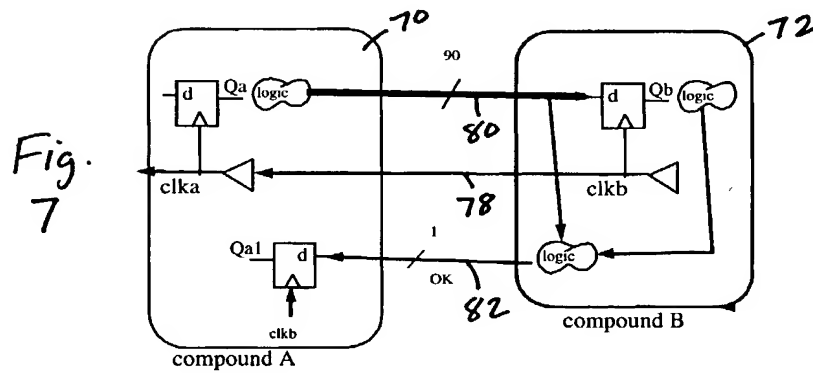
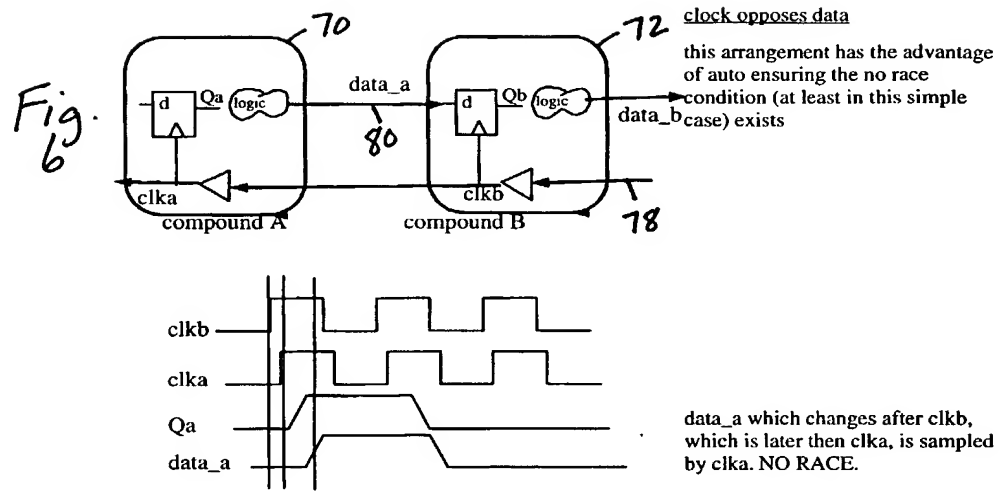


If the delay between "clk1" and "clk2" greater then the delay from Q to d of second flipflop, we have a race on our meaning right hand flipflop will sample the data of Q a whole clock period early.



clock runs with data

the problem is possible race. However, we control the logic on each flipflop leaving the compound, because it is always the same standart ring-interface module. we can ensure, that the delay will be at least enough. And more importantly easily checked after layout.



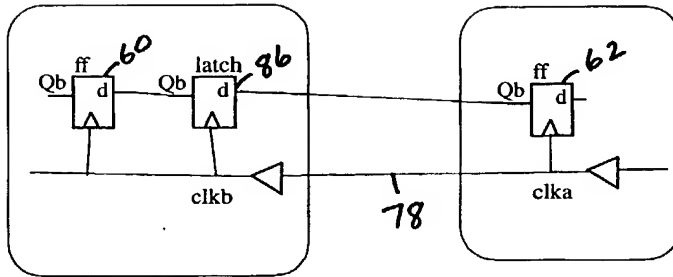


Fig. 8

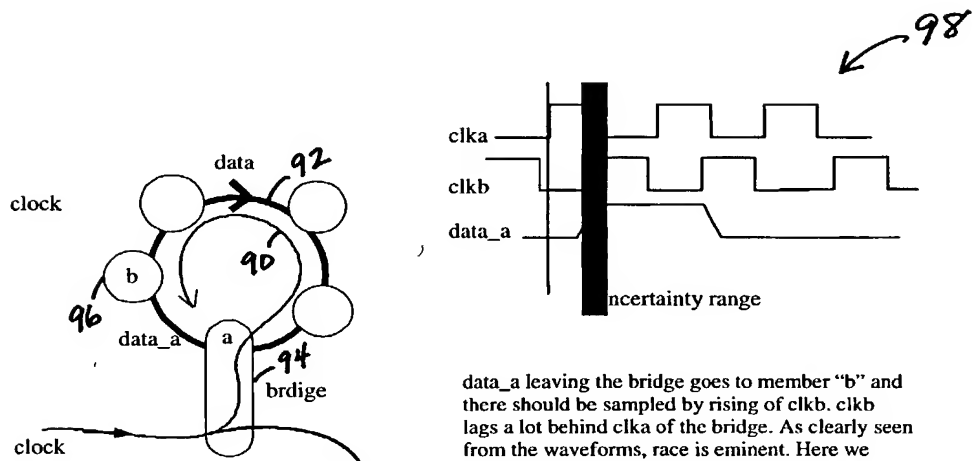


Fig. 9

data\_a leaving the bridge goes to member "b" and there should be sampled by rising of clkb. clkb lags a lot behind clka of the bridge. As clearly seen from the waveforms, race is eminent. Here we should add latches for all the data lines (~90). Adding latch works however if the delay between clka and clkb is less then 75% of cycle time. otherwise the uncertainty kills the usable time. It sets hard limit on the number of ring members. Also keep in mind that latches needed on each OK signal between members of the ring

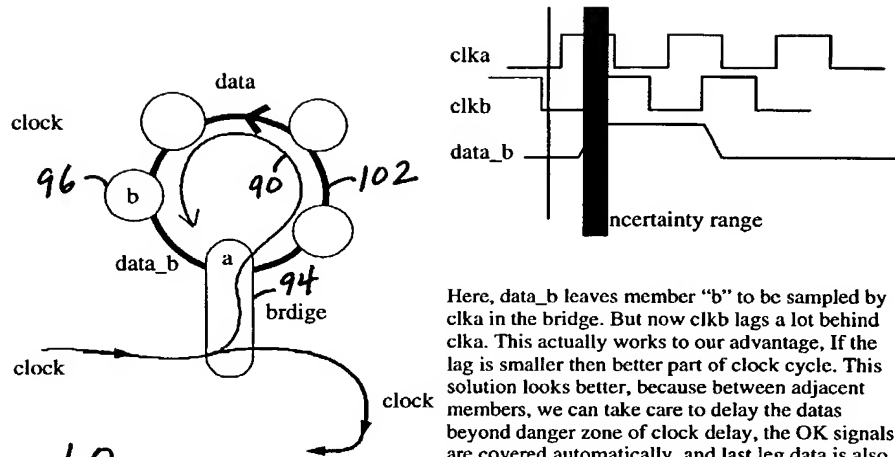
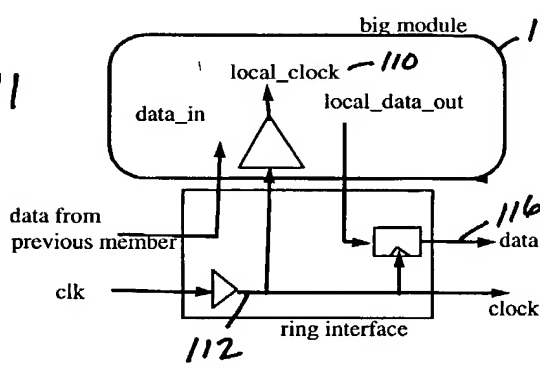


Fig. 10

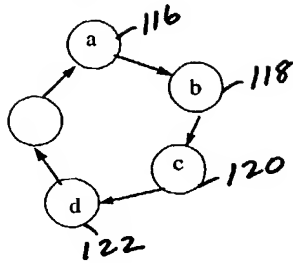
Here, data\_b leaves member "b" to be sampled by clka in the bridge. But now clkb lags a lot behind clka. This actually works to our advantage. If the lag is smaller than better part of clock cycle. This solution looks better, because between adjacent members, we can take care to delay the datas beyond danger zone of clock delay, the OK signals are covered automatically, and last leg data is also covered. The only signal not safe is the OK from bridge to "b" member. It will need a latch in "b".

Fig. 11



local clock lags behind ring\_interface clock of this module, because we presume the module is big. for data\_coming out, it is not a problem, it changes later then ring-i/f flipflops clock. However for data entering the module from previous member, the race is a possibility we must look into.

Fig. 12



if module "a" sends a message to module "b", ring works fine. However if most of the traffic is from "c" to "b", this is more expensive in terms of latency.

Another problem is "peak latency". Suppose that , "a" transmits mostly to "d" and "b" mostly to "c" In this case communication between "b" and "c" suffers degradation in case that peak traffic coincide.

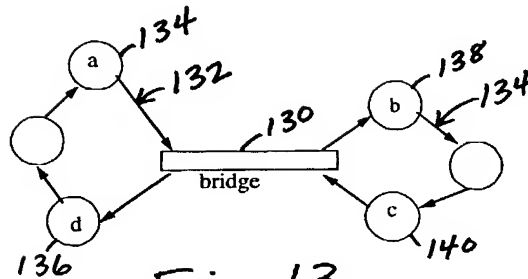
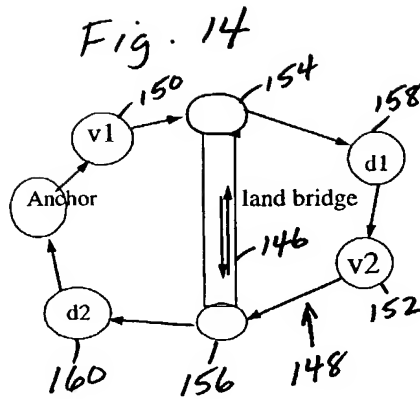
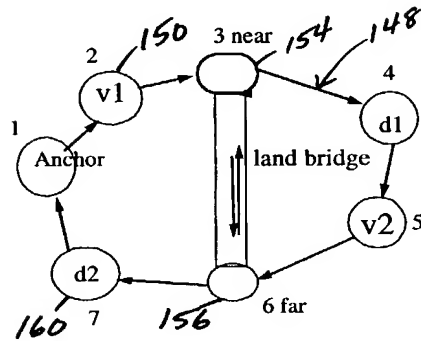


Fig. 13



Land bridge gets its name from the fact that it is a luxury. It spans across connected modules. The idea is simple. When V2 sends message to D1 it gets to one side of the bridge. This side analyzes the destination address and by some magic (explained later) decides to short-cut the path. The message re-appears at the other end of the bridge and gets fast to D1. By same magic, message from V1 to D2 get bypassed also. message from V1 to D1 is treated directly.

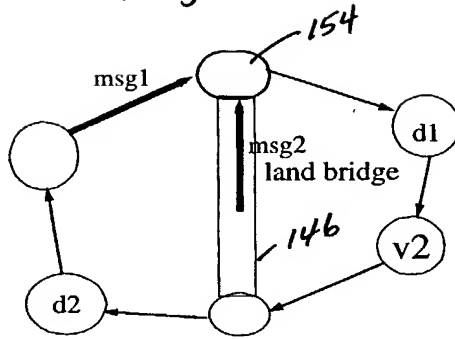


Enumeration is started by "Anchor" which assigns address=1 to itself. results of enumeration are labels 1 to 7. land bridge gets two addresses, as if it were not one module. there is "near" end, that got enumeration label "3", and the "far" end marked 6.

Fig. 15



Fig. 16



msg1 and msg2 arrive at the same time.  
the bridge end must make a decision  
which message to forward first.

It can be shown that unwise decision can  
lead to freezout, deadlock and option price  
dropping to 5\$.

Therefore MSG2 gets the priority.





Fig. 20

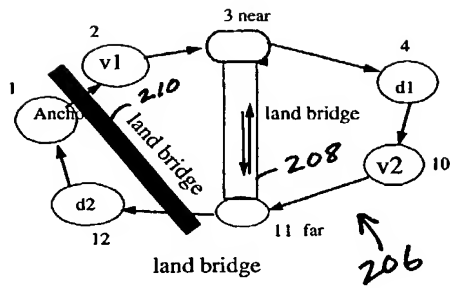


Fig. 21

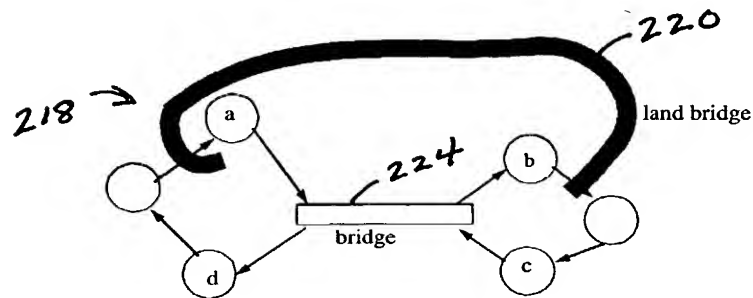
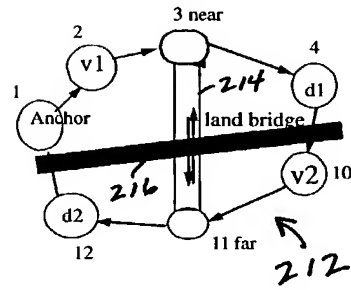
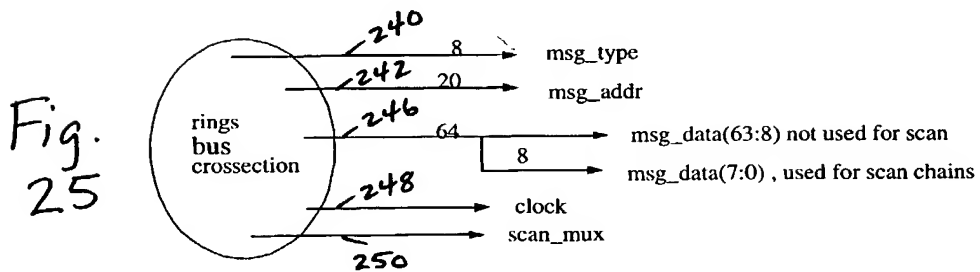
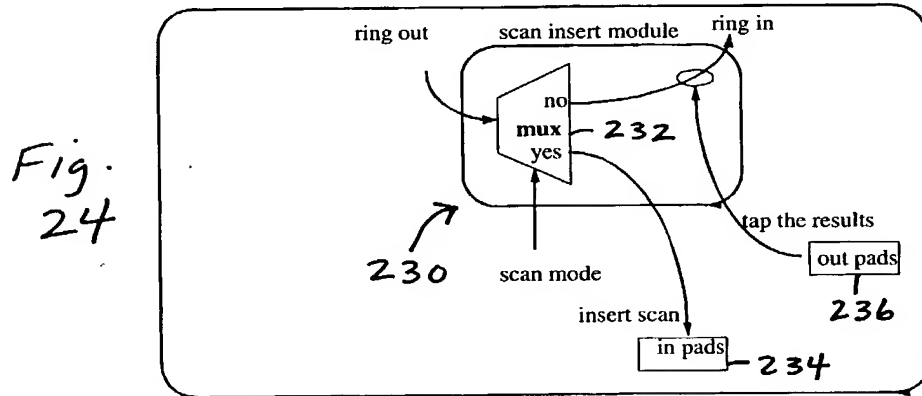
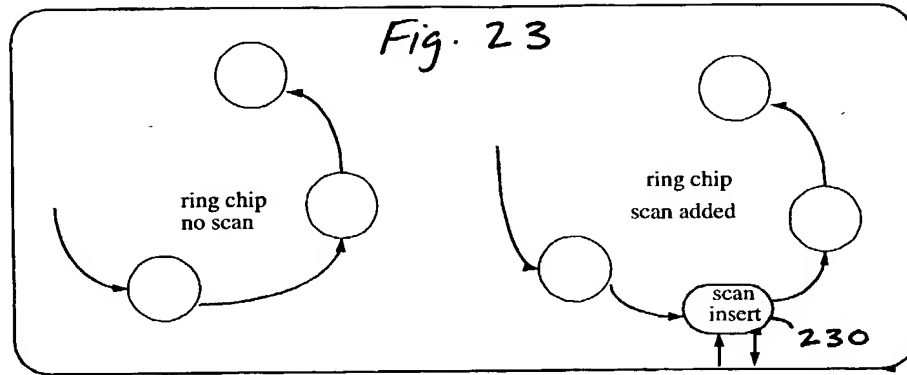
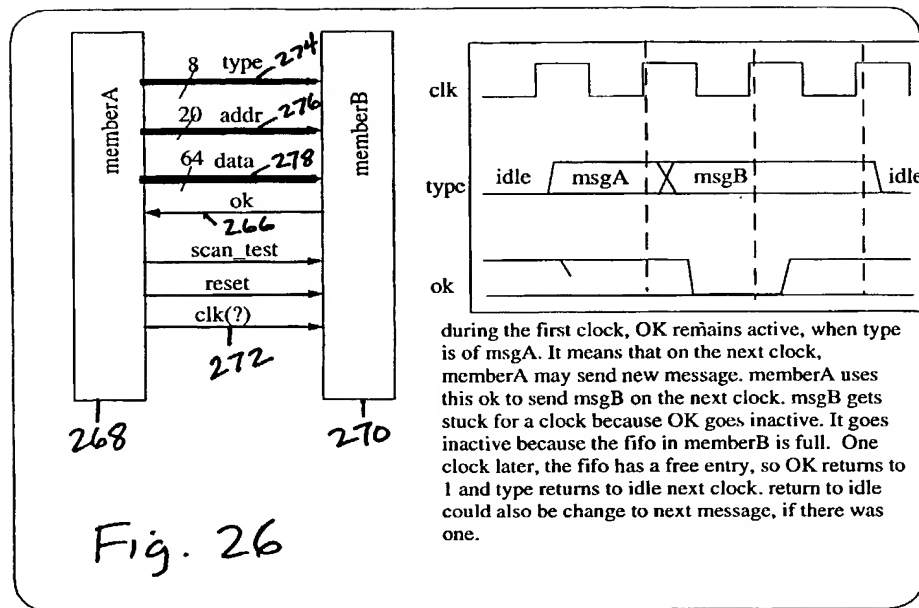


Fig. 22





during the first clock, OK remains active, when type is of msgA. It means that on the next clock, memberA may send new message. memberA uses this ok to send msgB on the next clock. msgB gets stuck for a clock because OK goes inactive. It goes inactive because the fifo in memberB is full. One clock later, the fifo has a free entry, so OK returns to 1 and type returns to idle next clock. return to idle could also be change to next message, if there was one.

Fig. 27

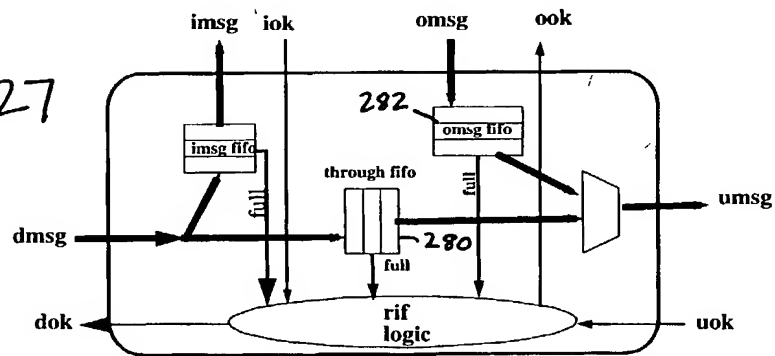
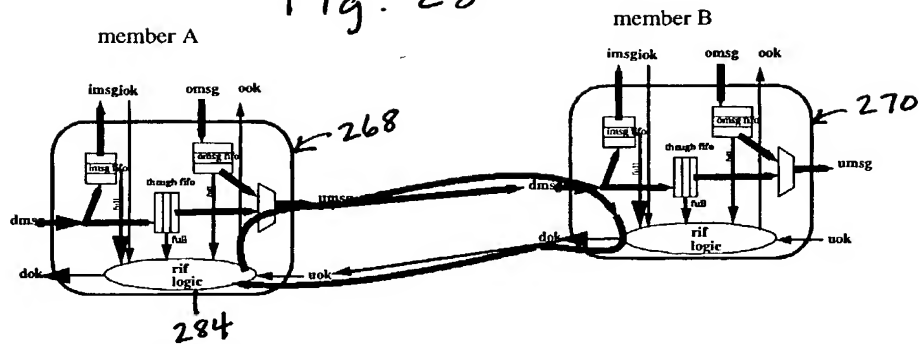


Fig. 28



The incoming messages are examined first to see if it is supervisor or work/program. Work/program messages have address field. We check if it is our address. Since we know that our address is aligned to our power of 2, The address mask (named split mask) causes only certain number of upper bits to be compared. The lower part of the address is passed inside as internal address. The upper bits are compared against self-address register. This register gets its value during enumeration protocol. The lower part of this register is always masked. Hopefully synthesis will delete the unused bits implementation.

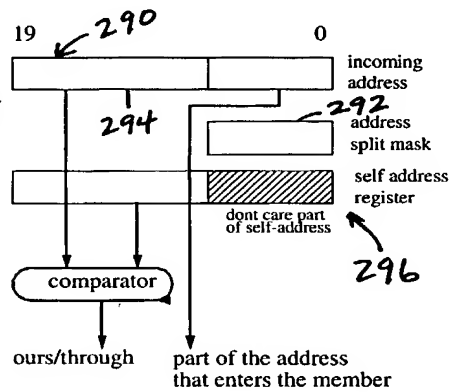


Fig. 29

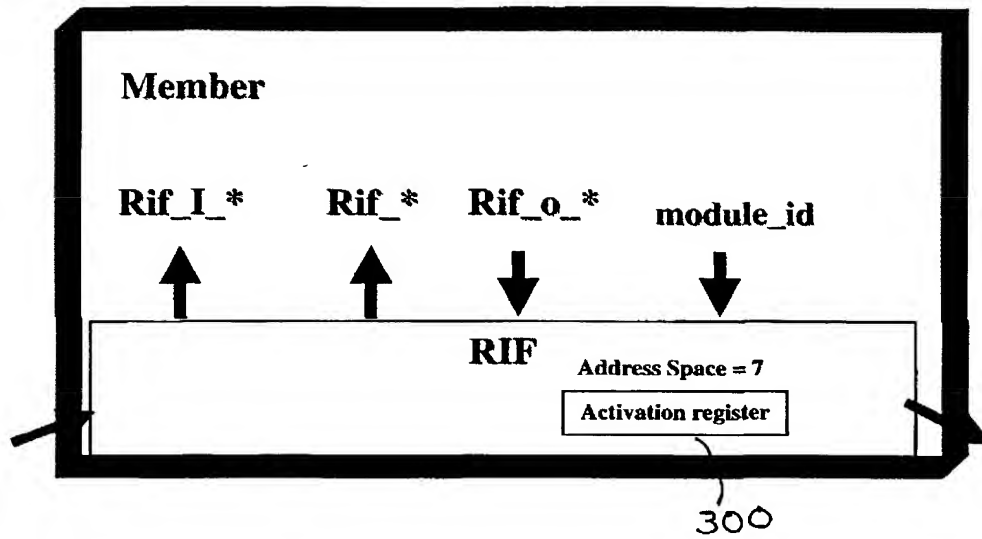
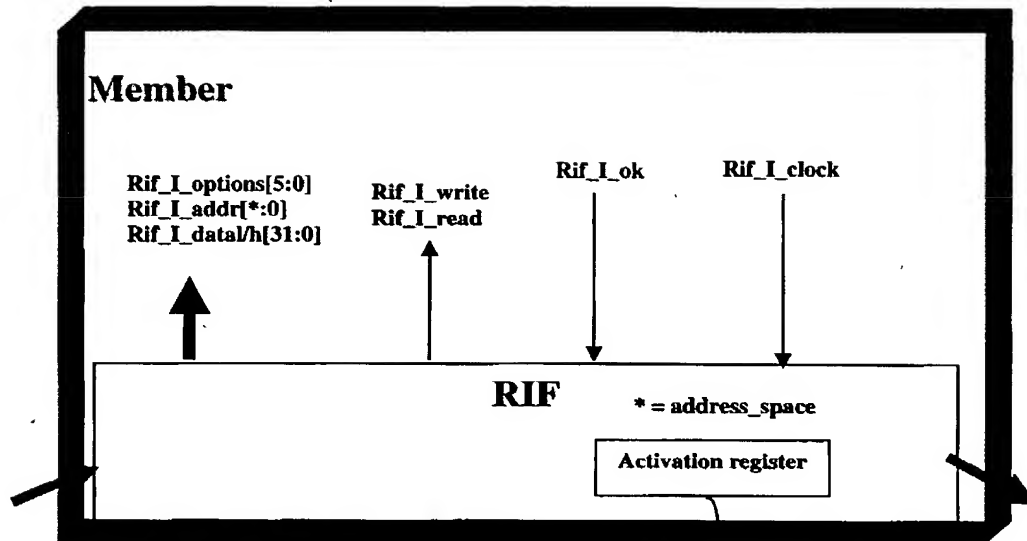


Fig. 30





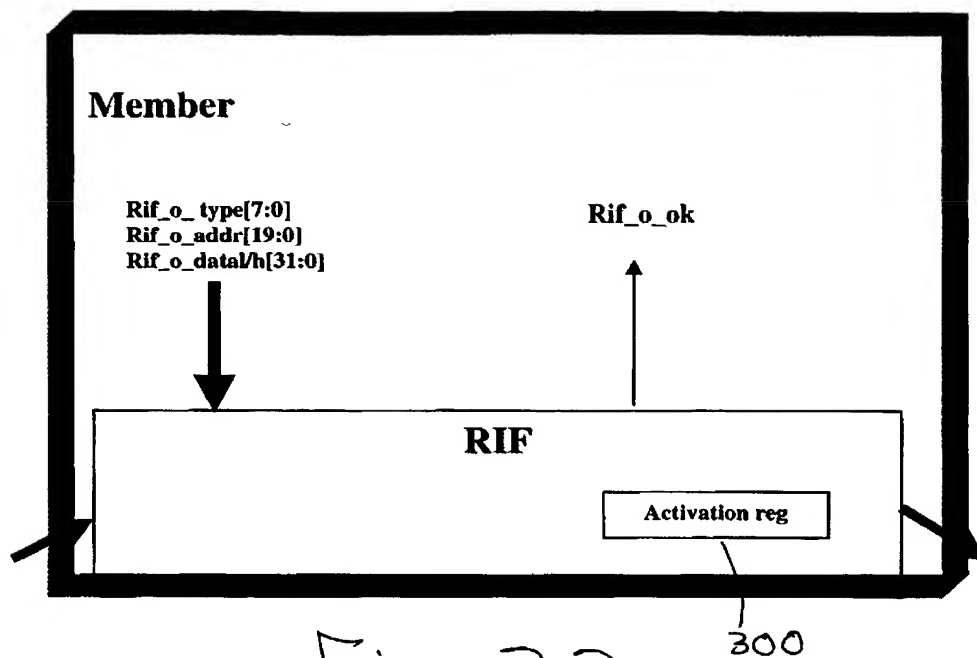
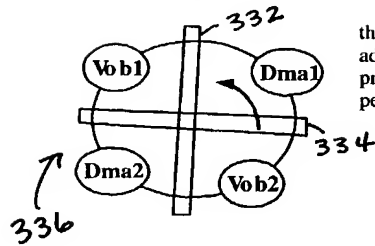


Fig. 32

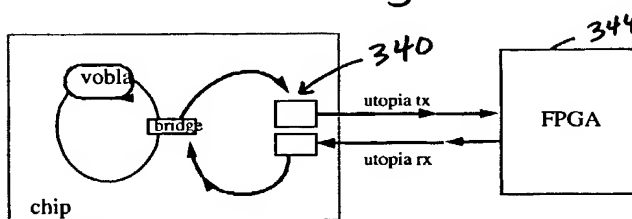
Fig. 33

Fig. 34



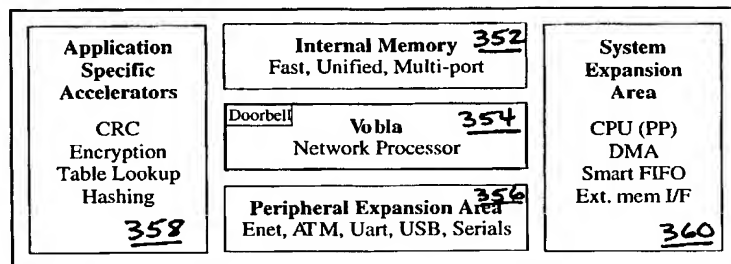
the second land bridge solves most traffic problems, but adds 4 clocks in the overall ring length. This is not a big problem because no message should travel the whole perimeter.

Fig. 35



The utopia interface is forced into mode that communicates in messages, not cells. We using the I/O and maybe some of the logic.

Fig. 36



350

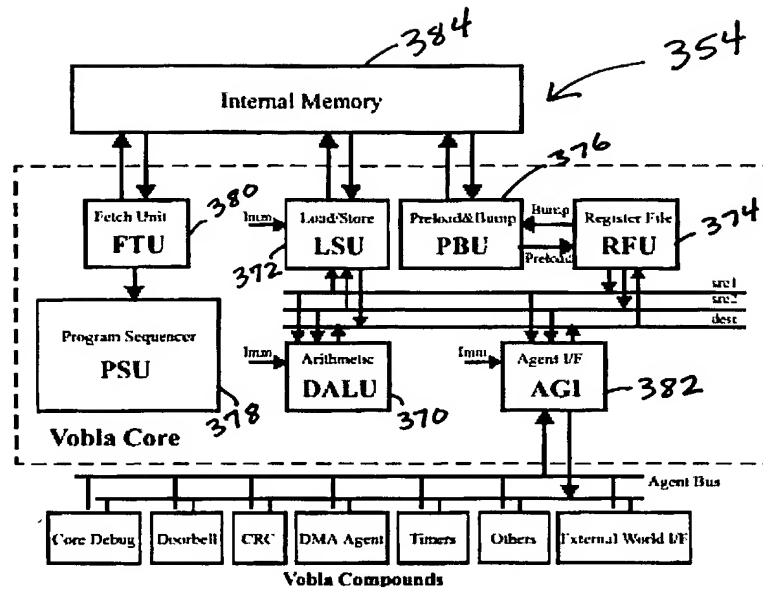
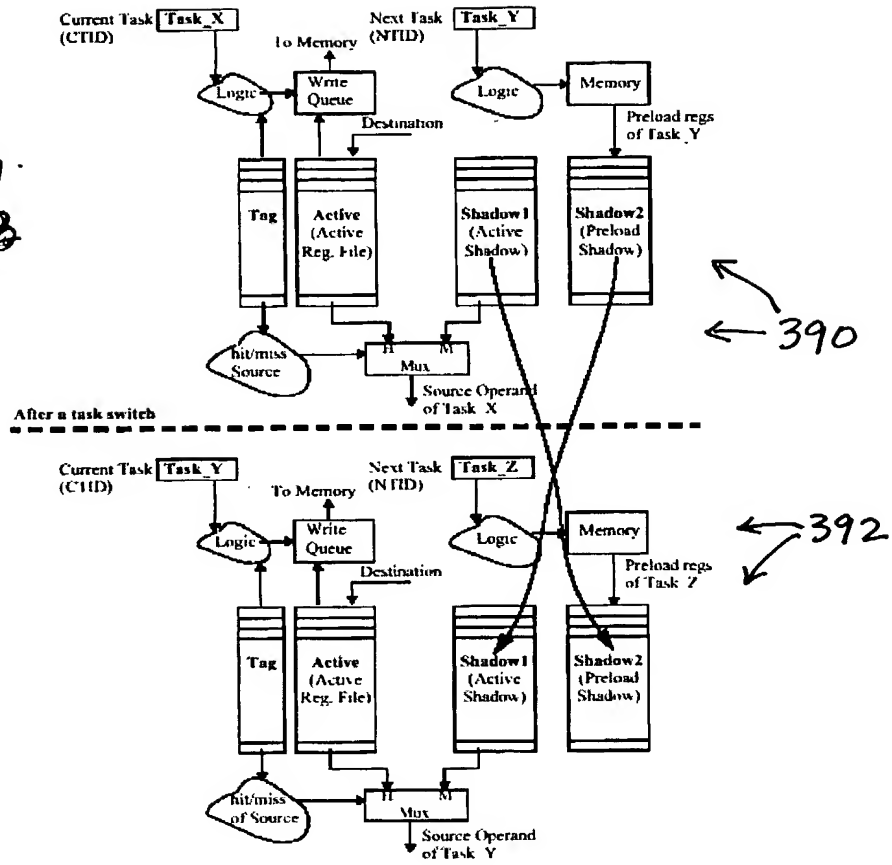


Fig.  
37

Fig.  
38



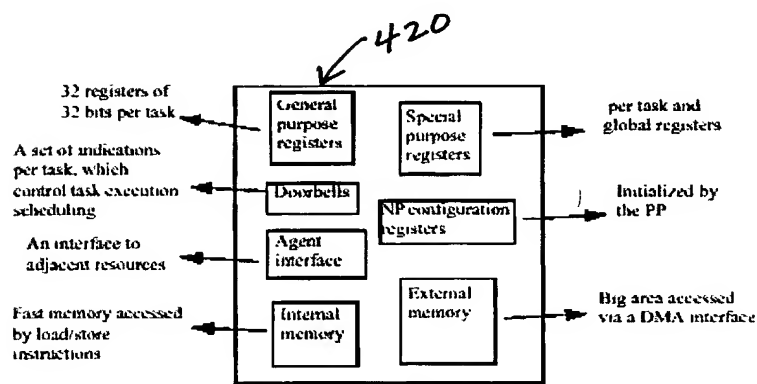
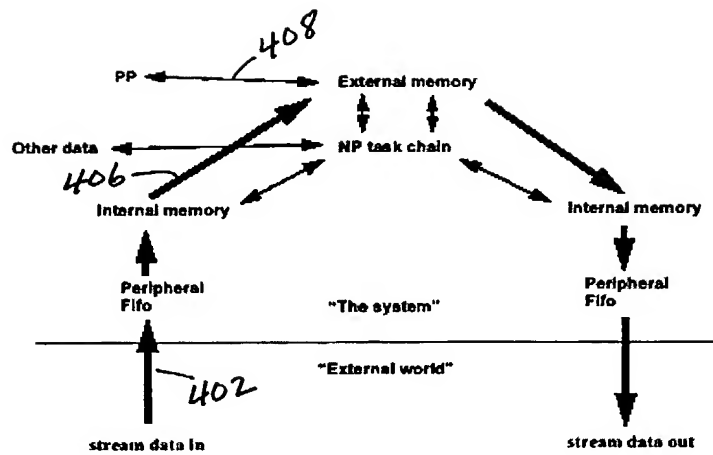
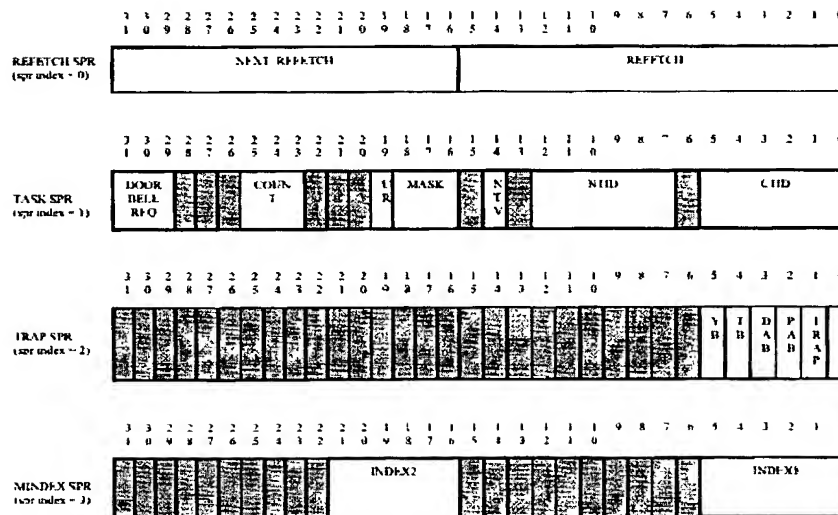


Fig. 41

R1 register:



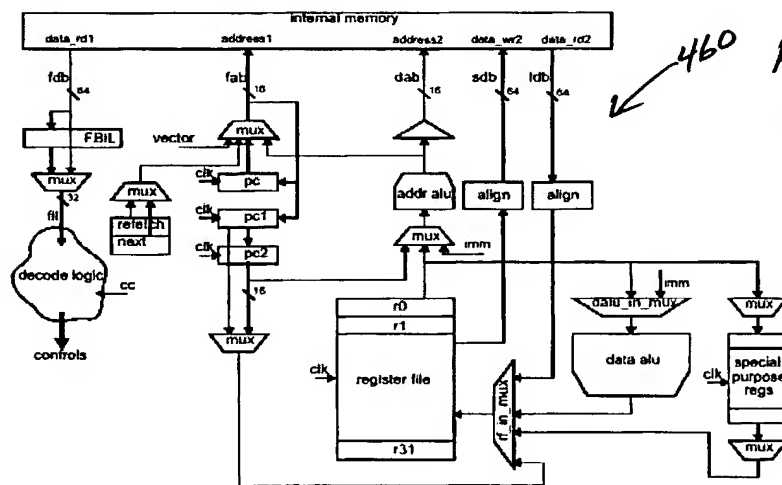
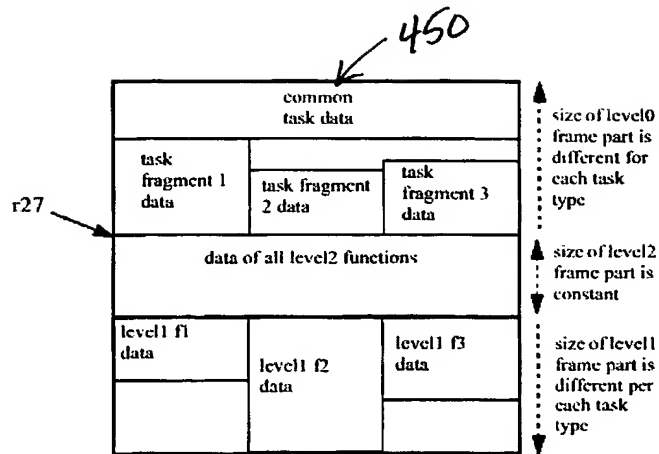
s - sticky bit  
 eq - equal/zero  
 lt - less than/negative  
 gt - greater than/positive  
 c - carry  
 mb - reflection of the RAM multi-reader busy indication.





Frame structure  
of an example  
task type

Fig.  
43



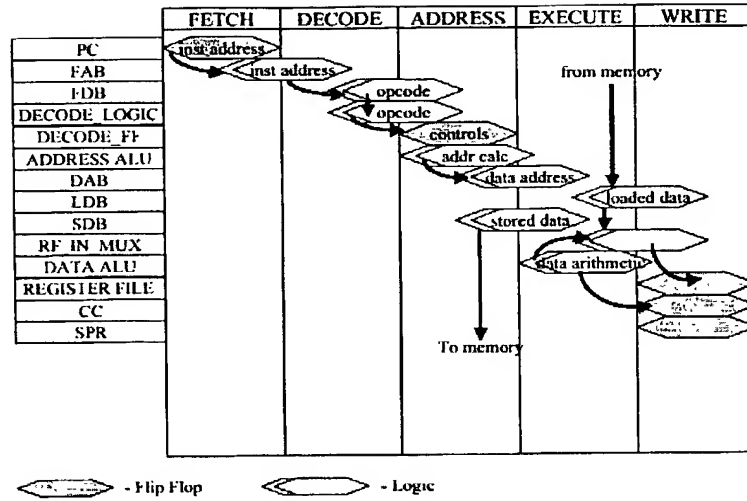


Fig. 45

Fig. 46

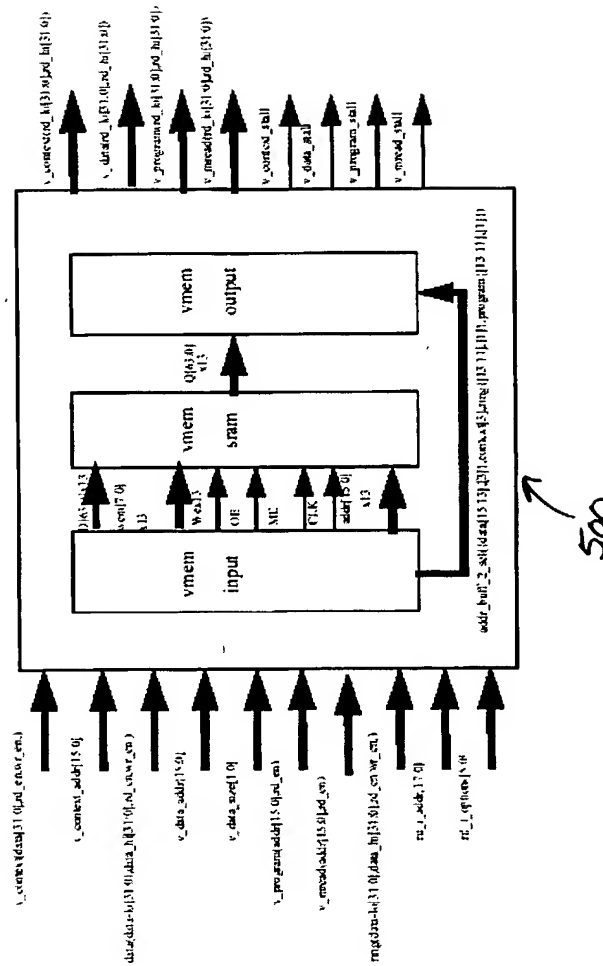
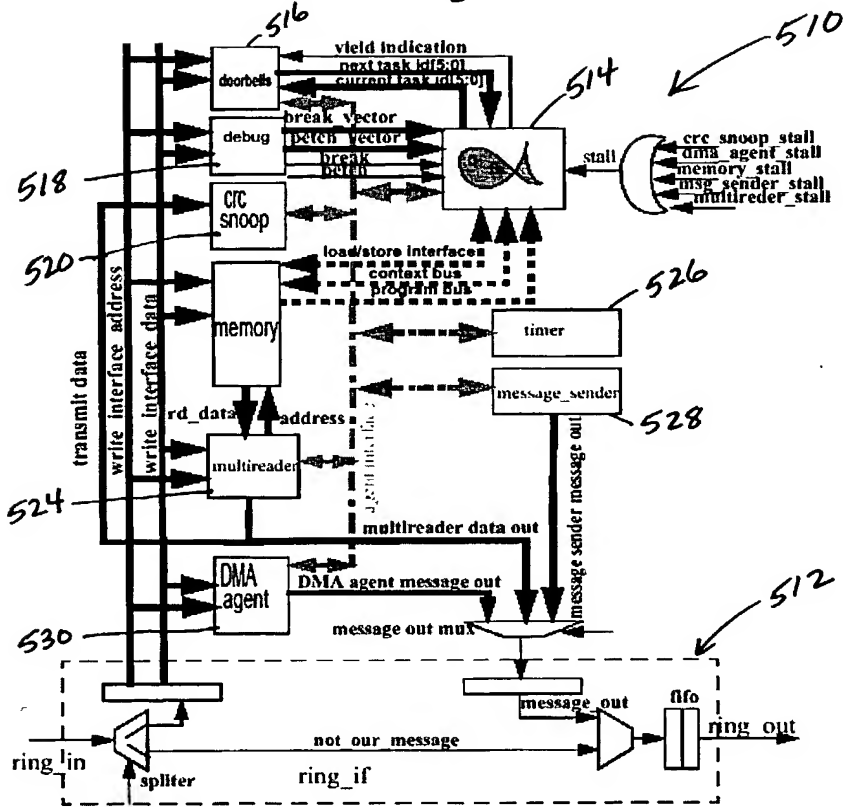


Fig. 47



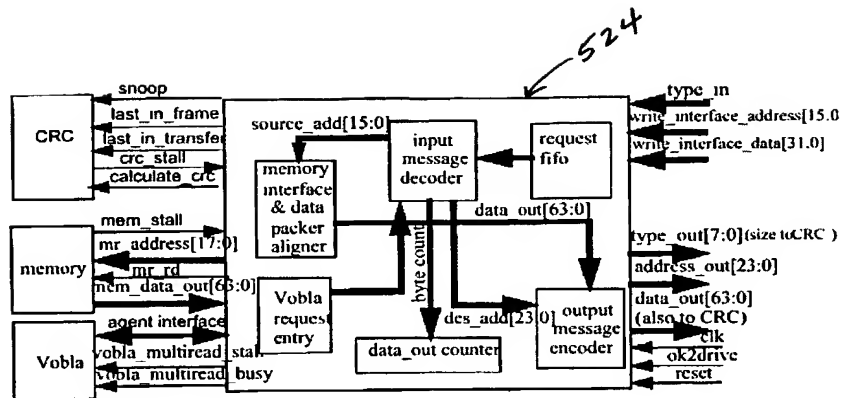


Fig. 48

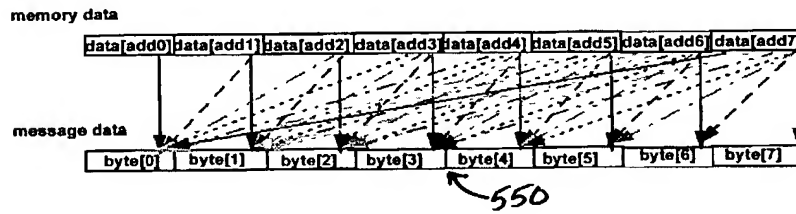


Fig. 49

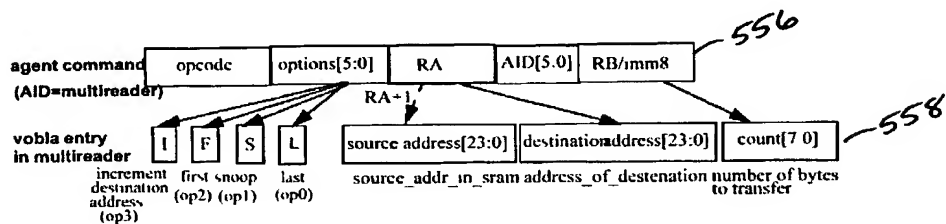


Fig. 50

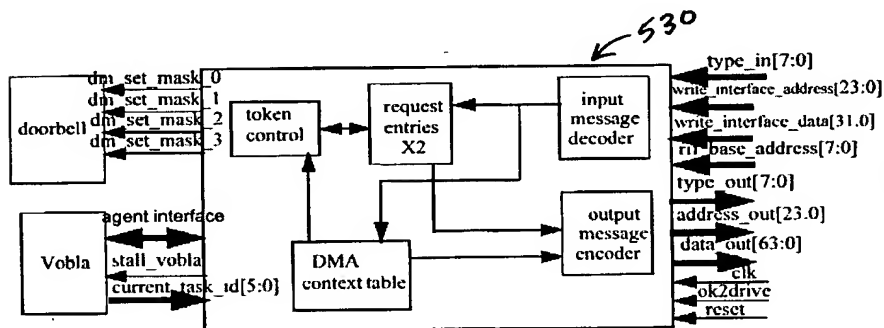
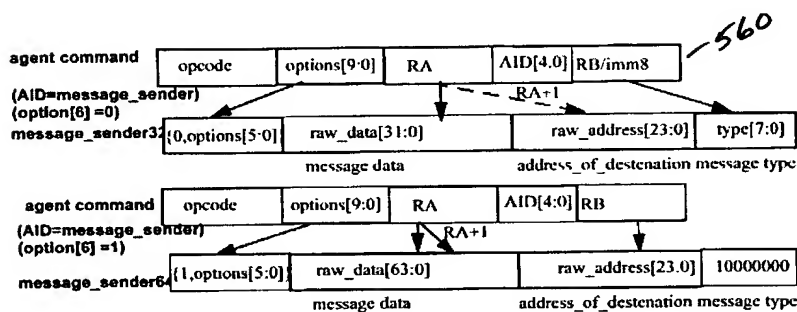
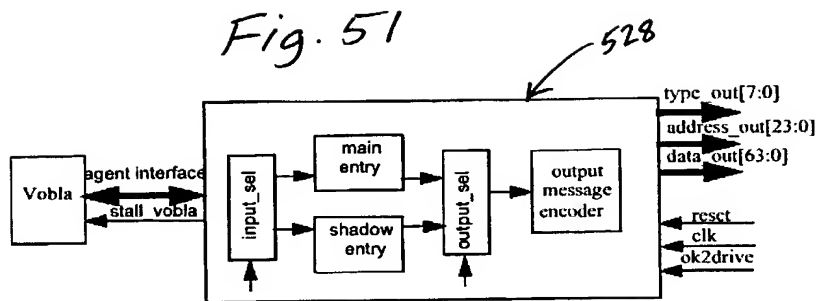


Fig. 54

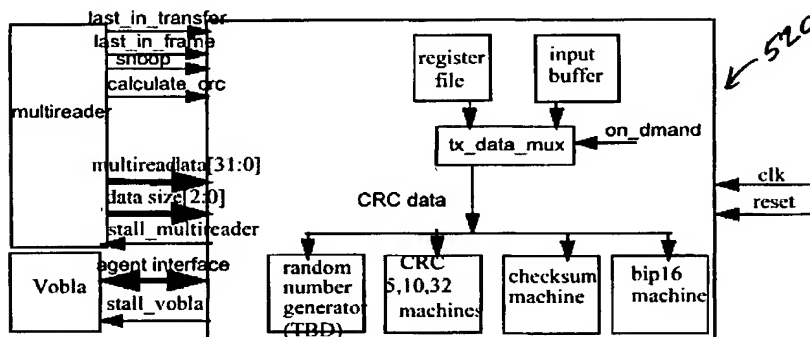
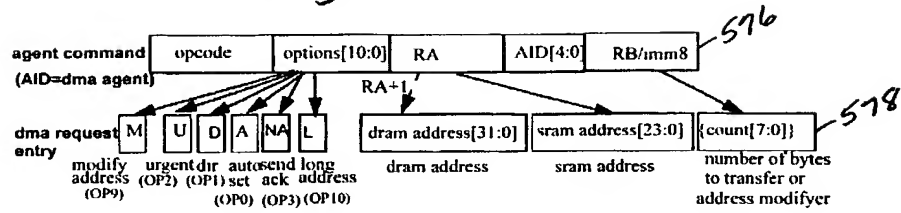
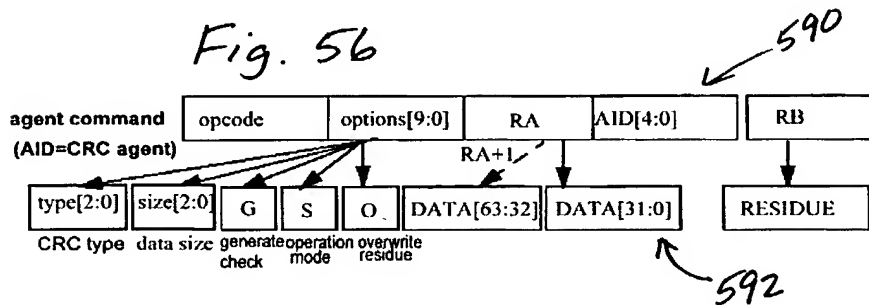


Fig. 55

Fig. 56



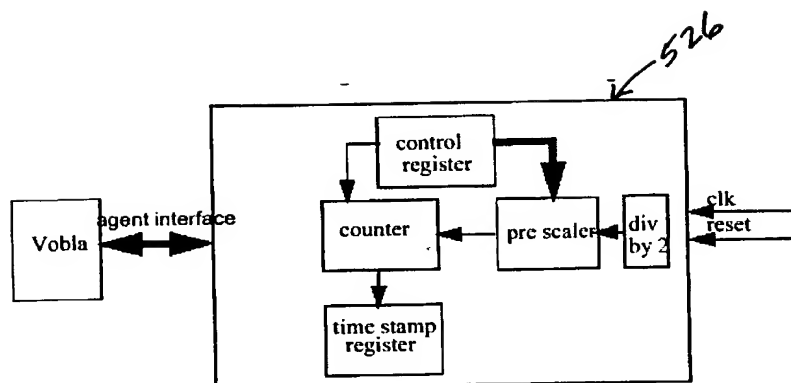


Fig.  
57

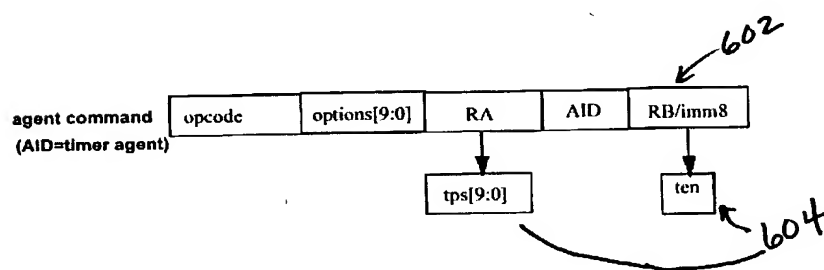


Fig.  
58



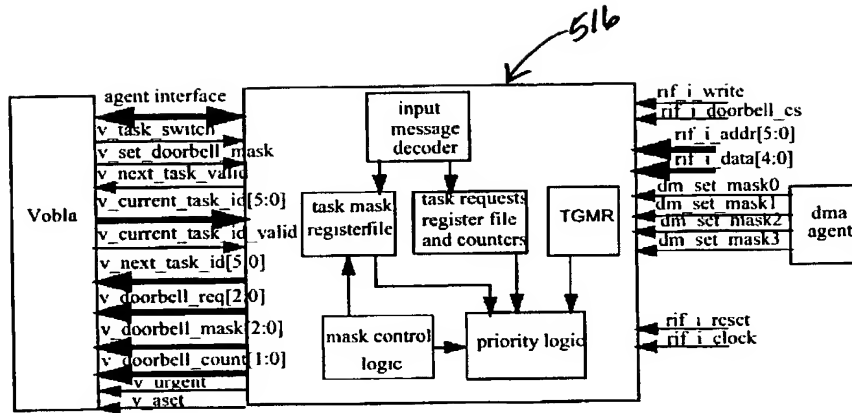


Fig. 59

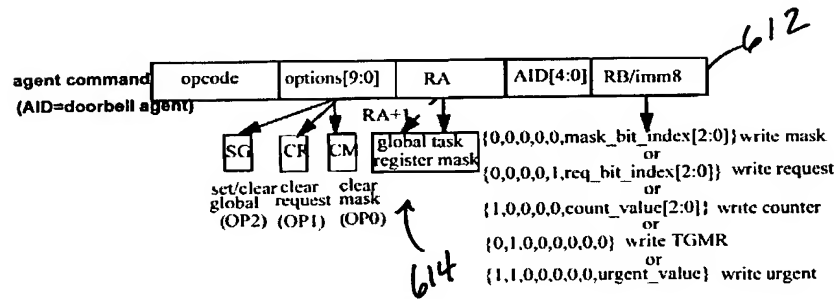
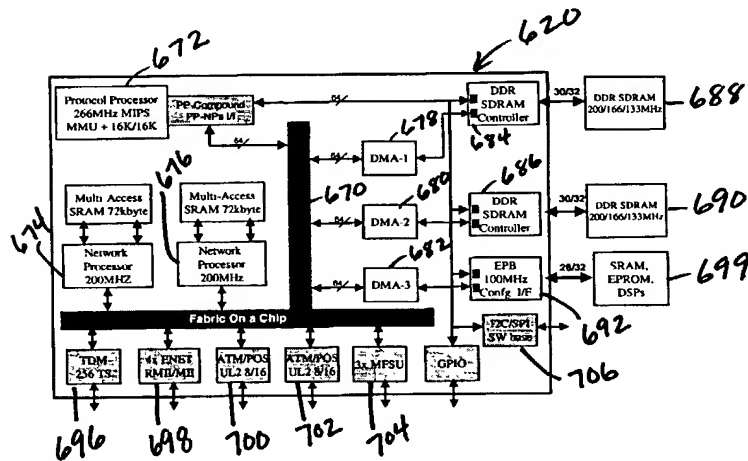
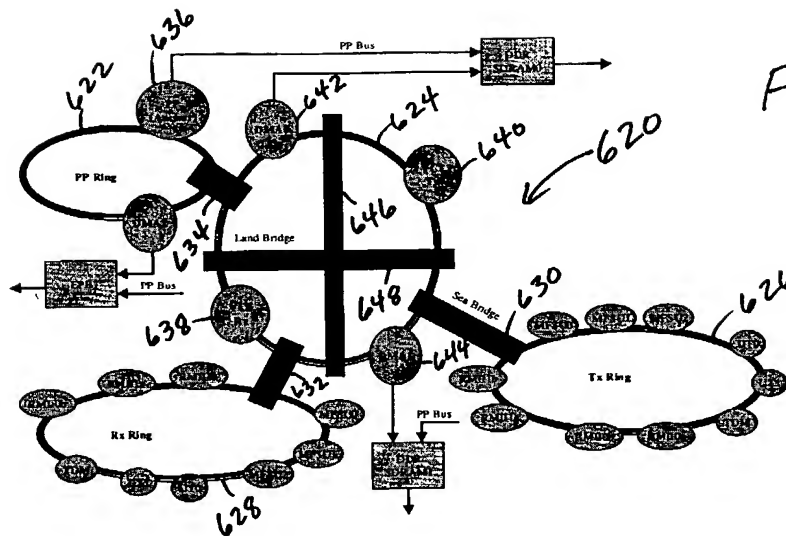
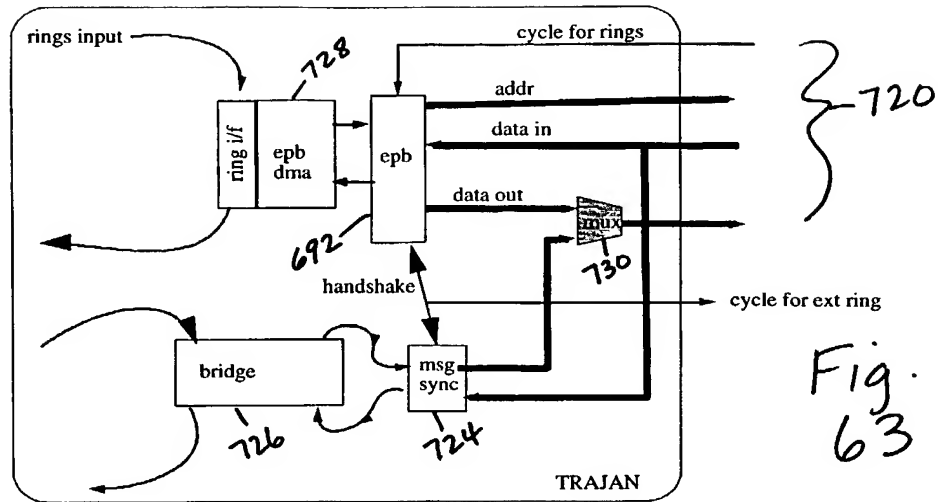


Fig. 60





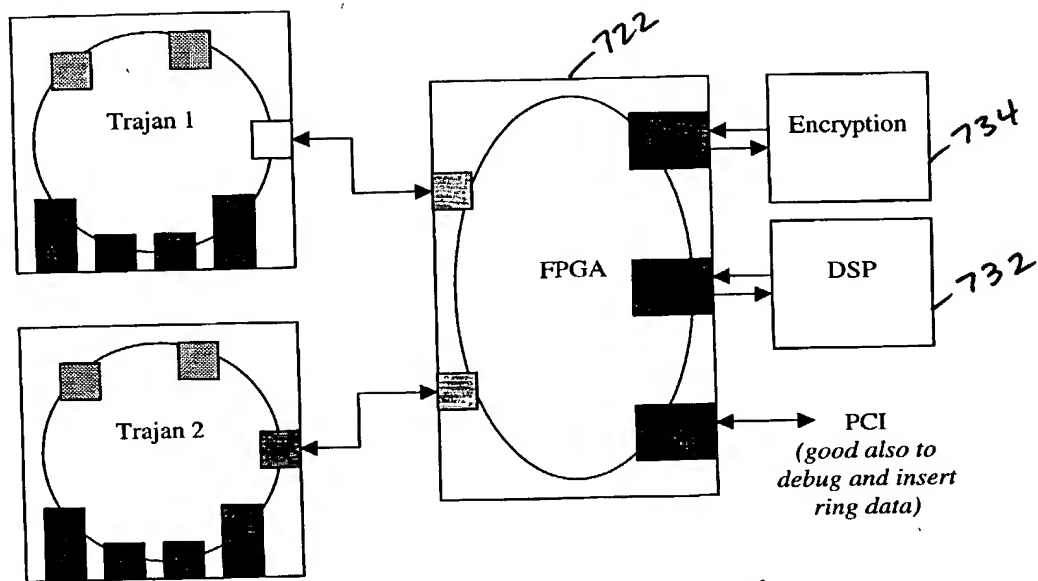


Fig. 64

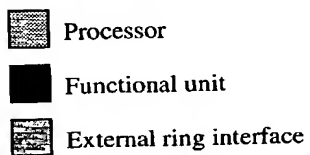


Fig. 65

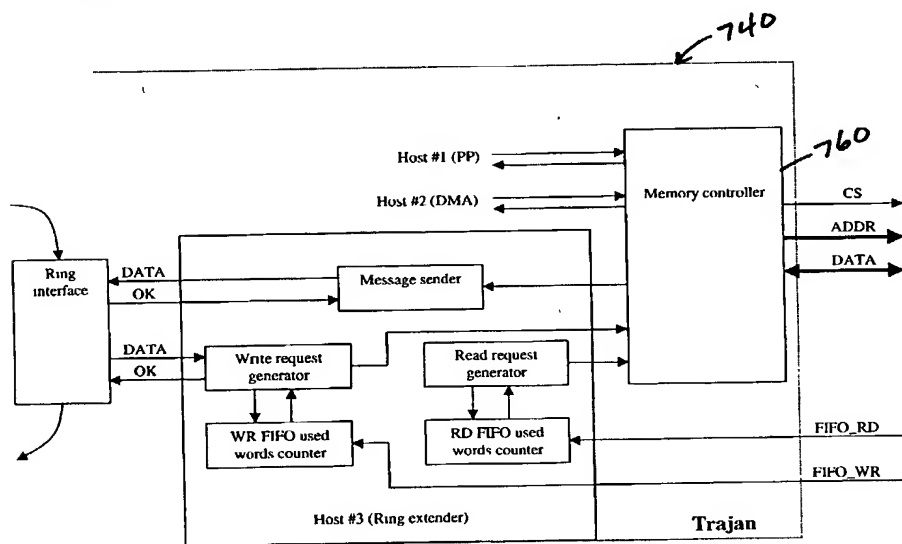
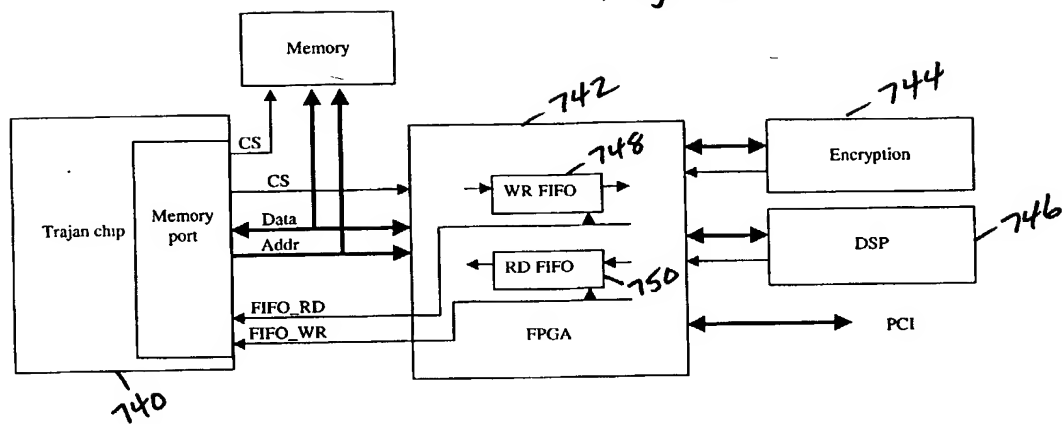


Fig. 66

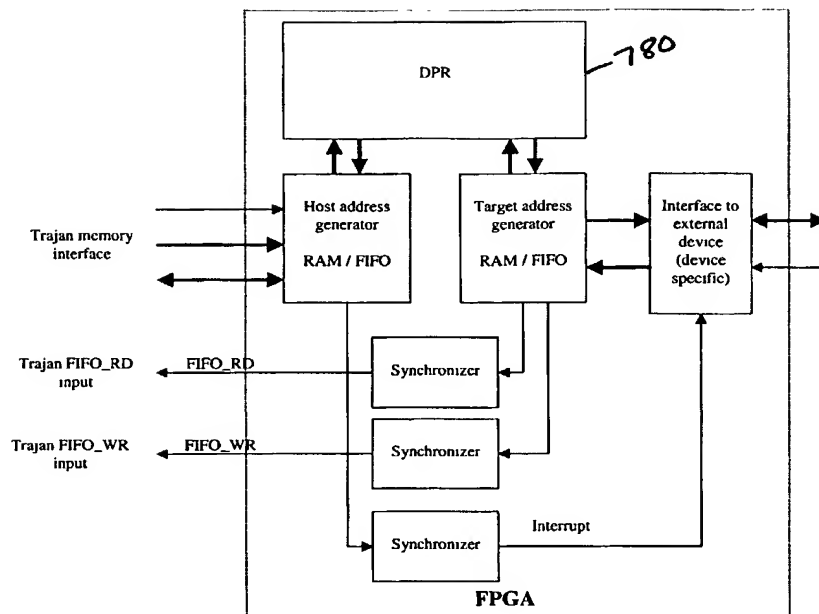


Fig. 67

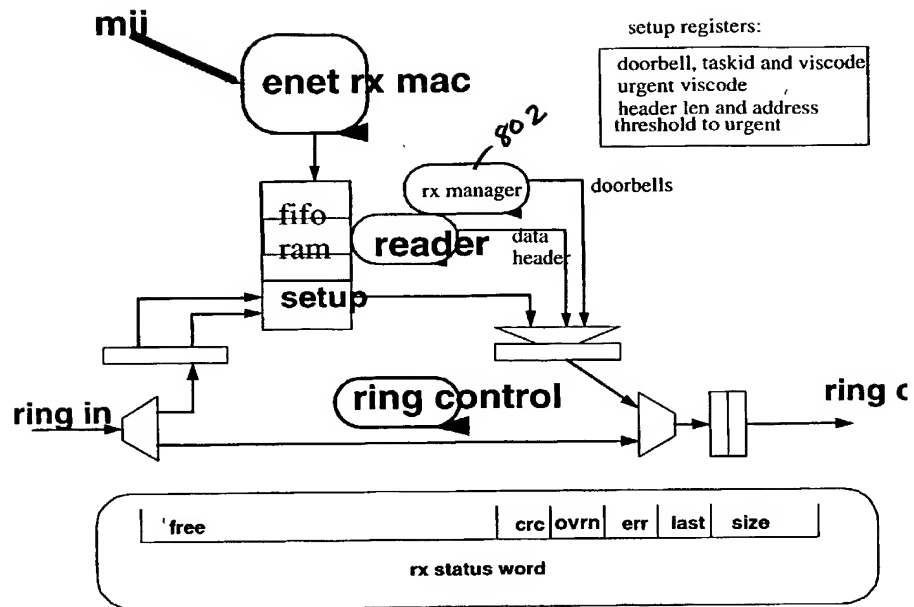


Fig. 68

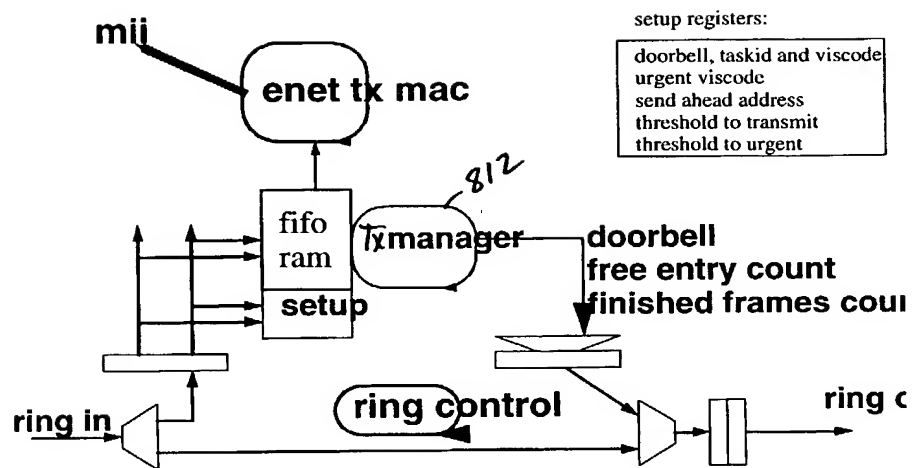


Fig. 69



# Control Plane

Signaling Protocols  
Protocol Management  
Exception Handling  
System Control &  
Configuration

# Data Plane

Per/packet handling  
Forwarding Decision  
Classification  
QoS Handling  
Queueing  
Scheduling  
Formatting

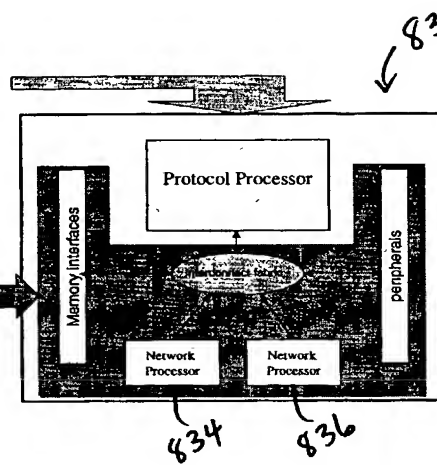


Fig. 70

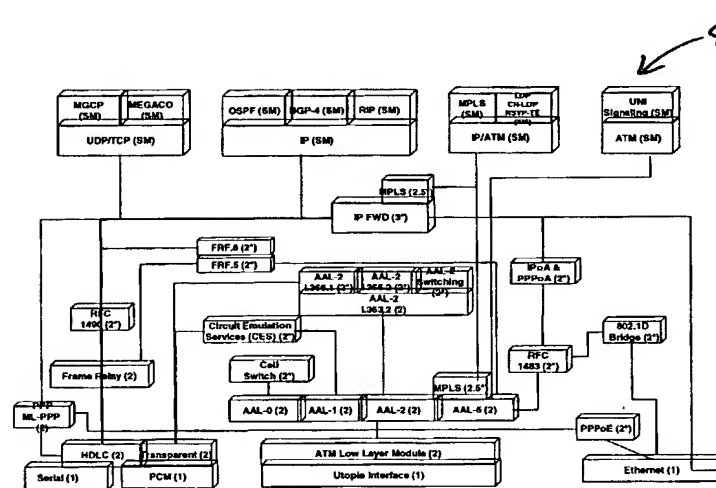


Fig. 71

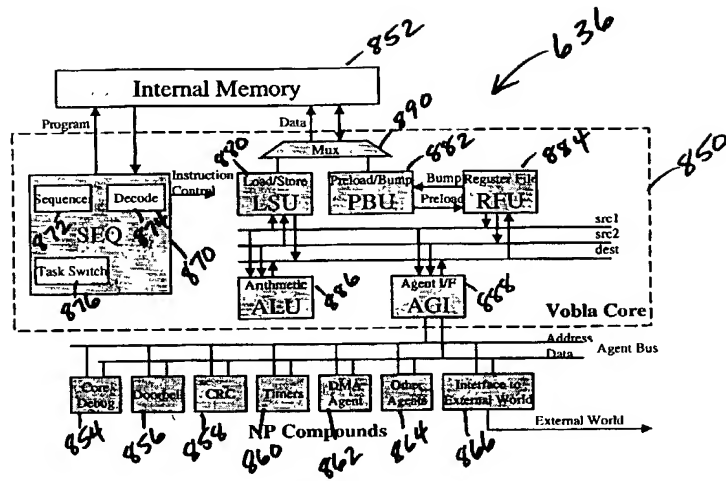


Fig. 72

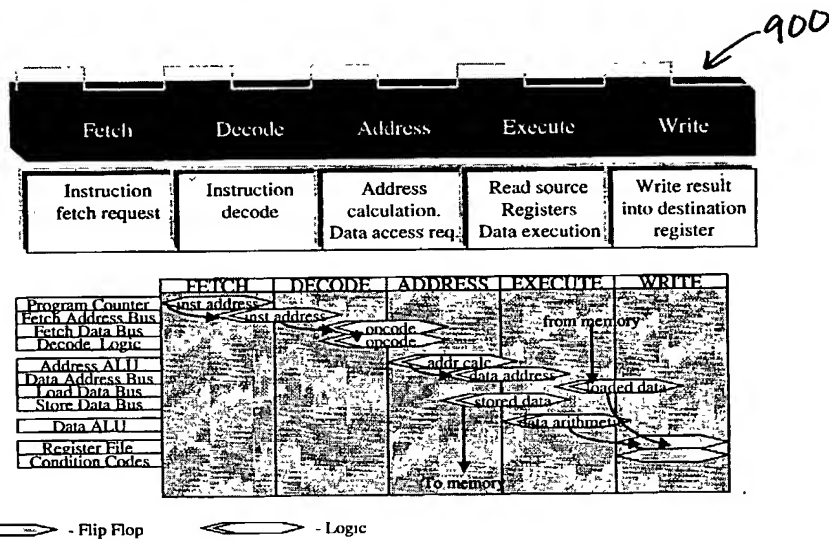
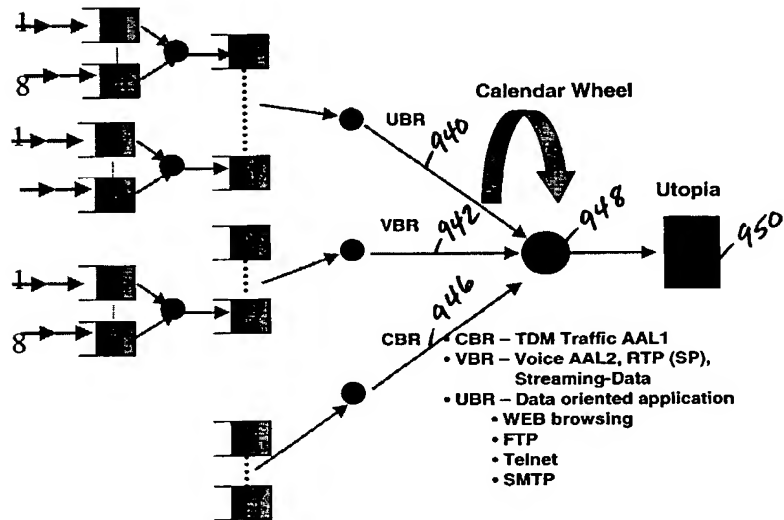
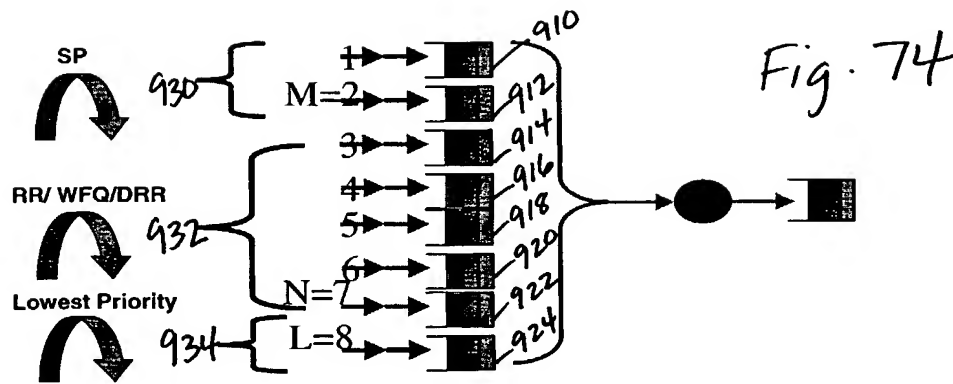


Fig. 73



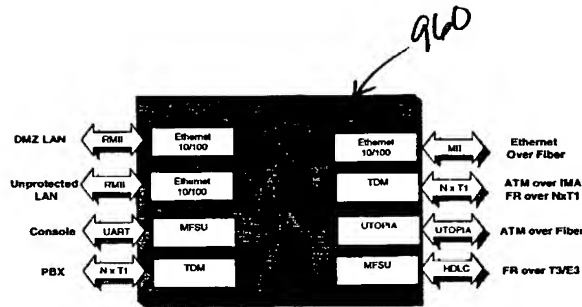


Fig.  
76

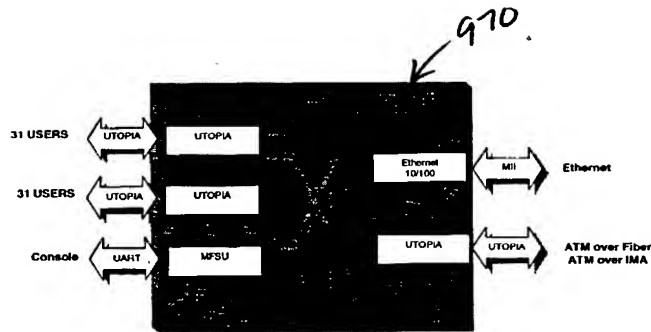


Fig.  
77

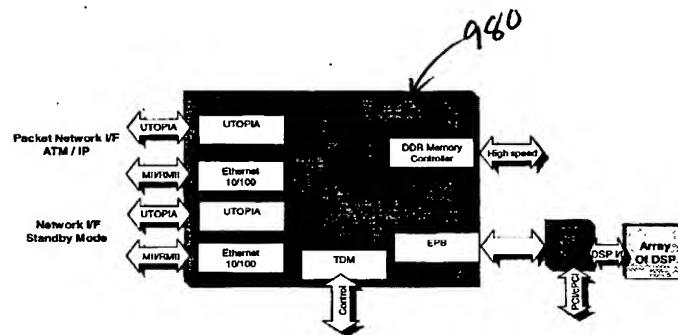


Fig.  
78

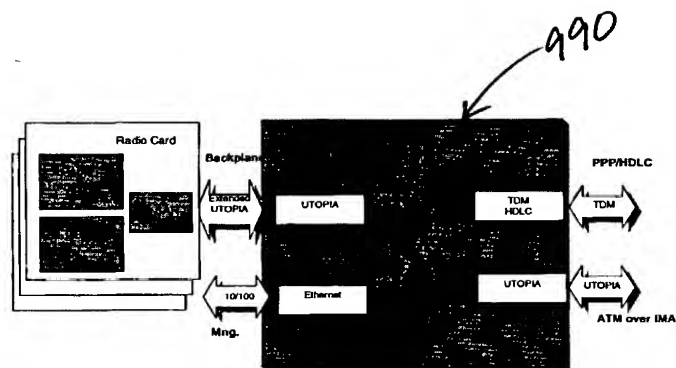


Fig.  
79

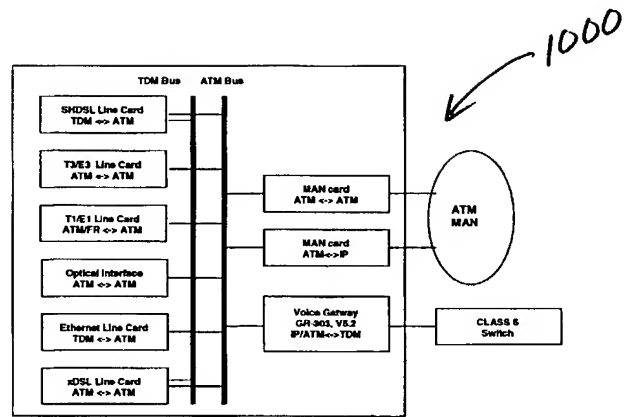


Fig. 80

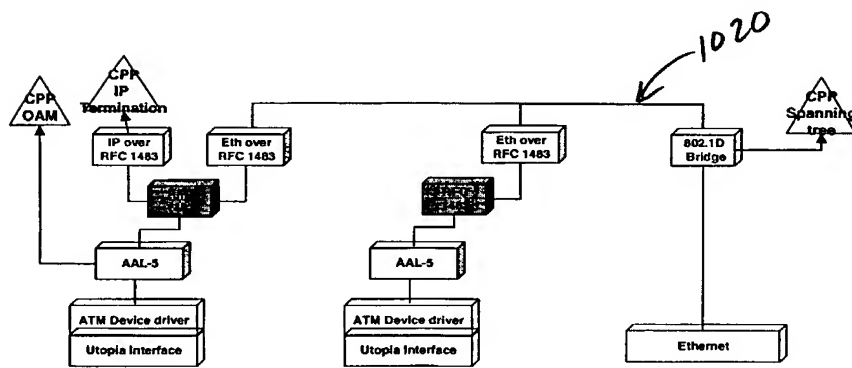
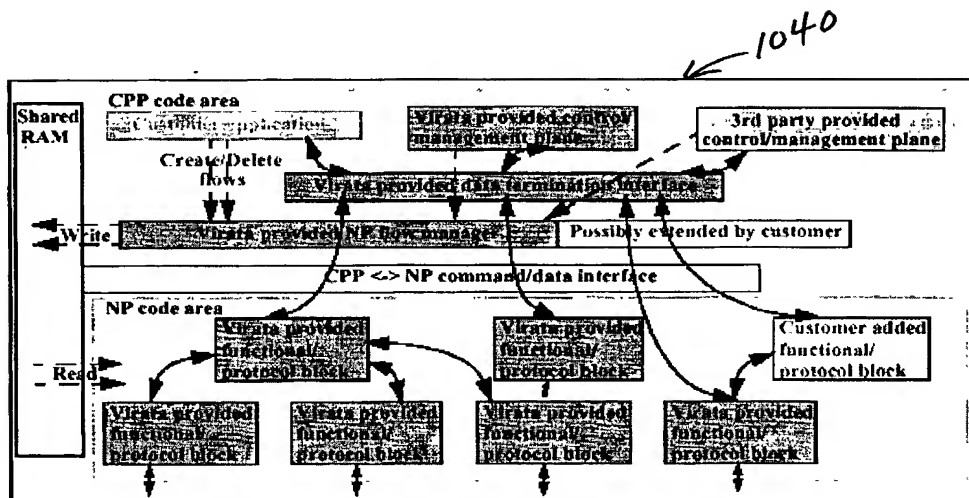
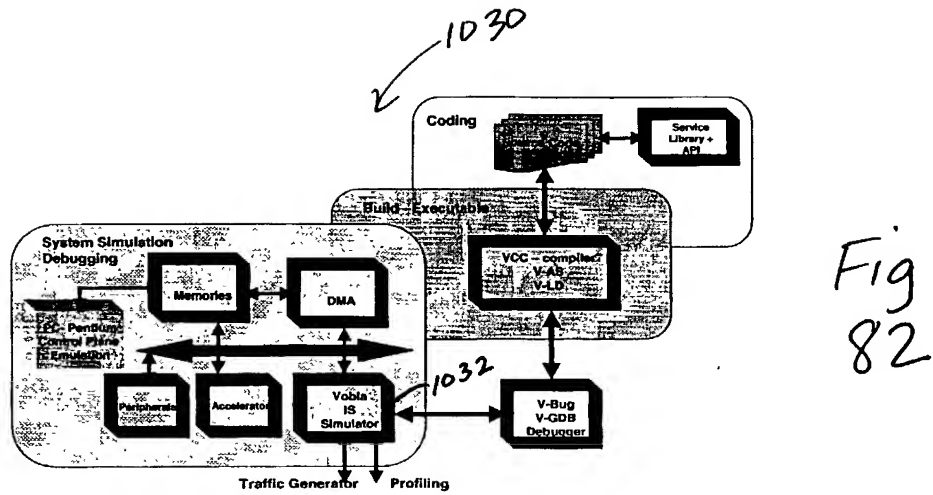


Fig. 81



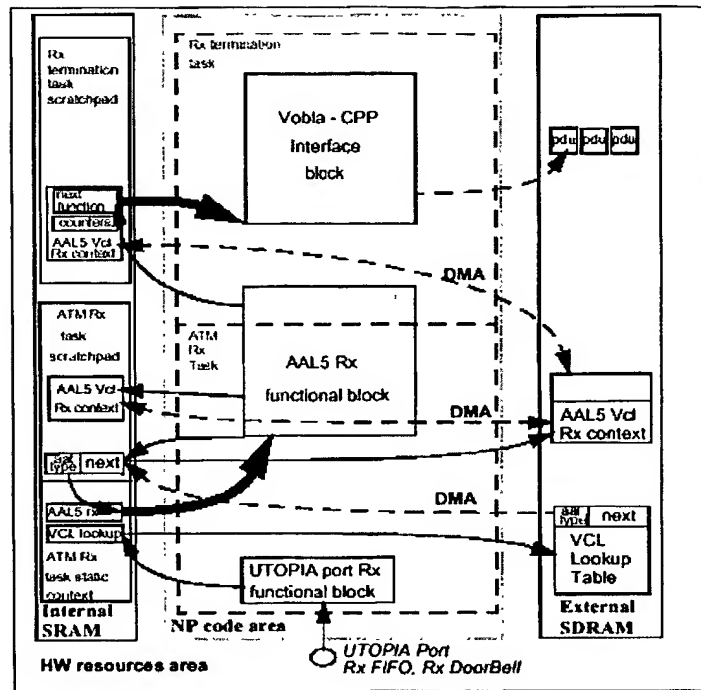


Fig. 84



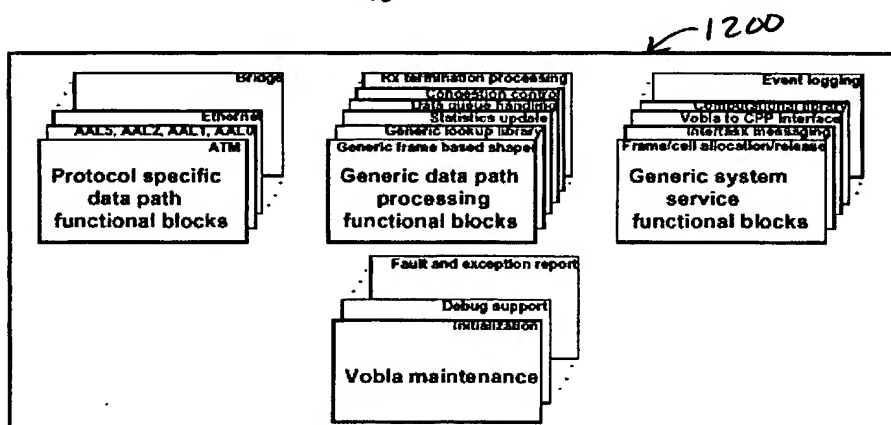
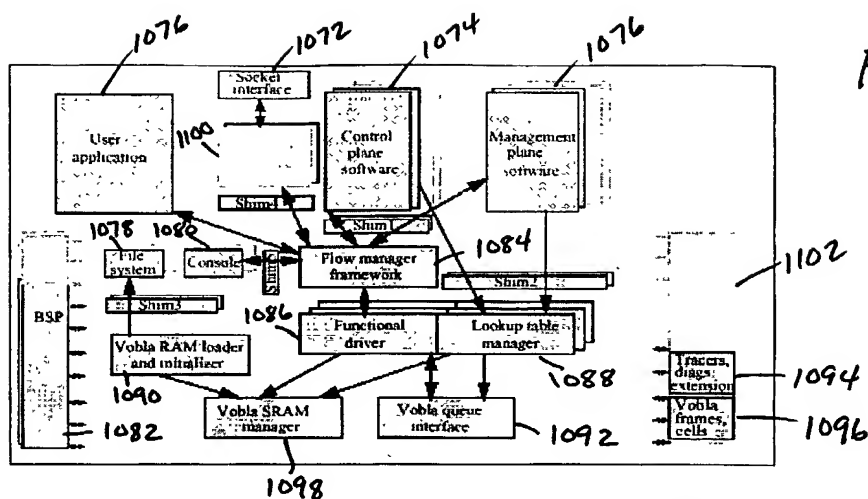


Fig. 86

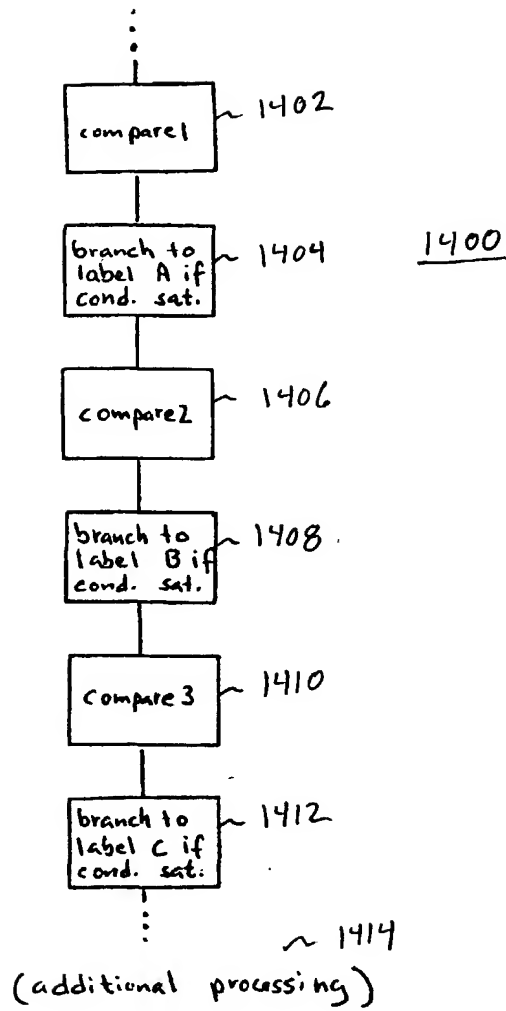


FIG. 87  
(PRIOR ART)

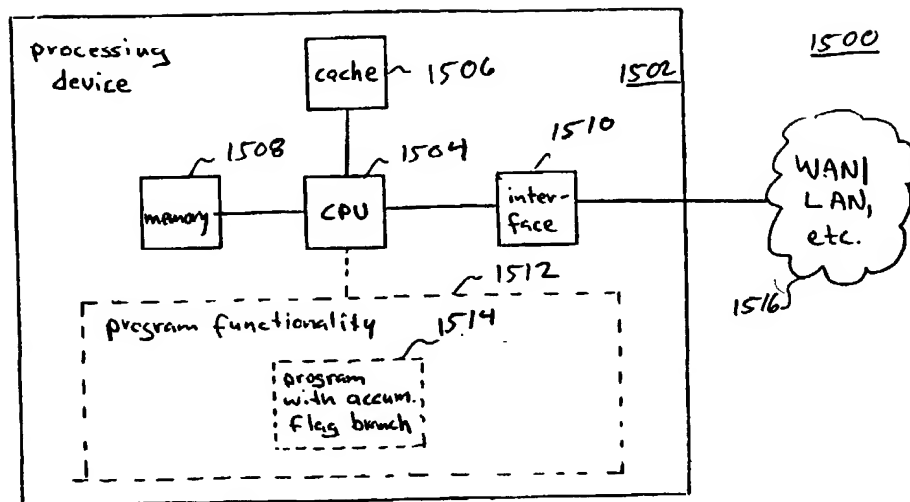


FIG. 88

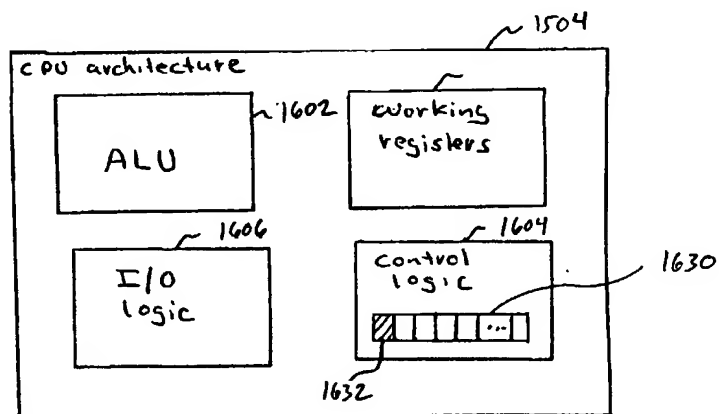


FIG. 89

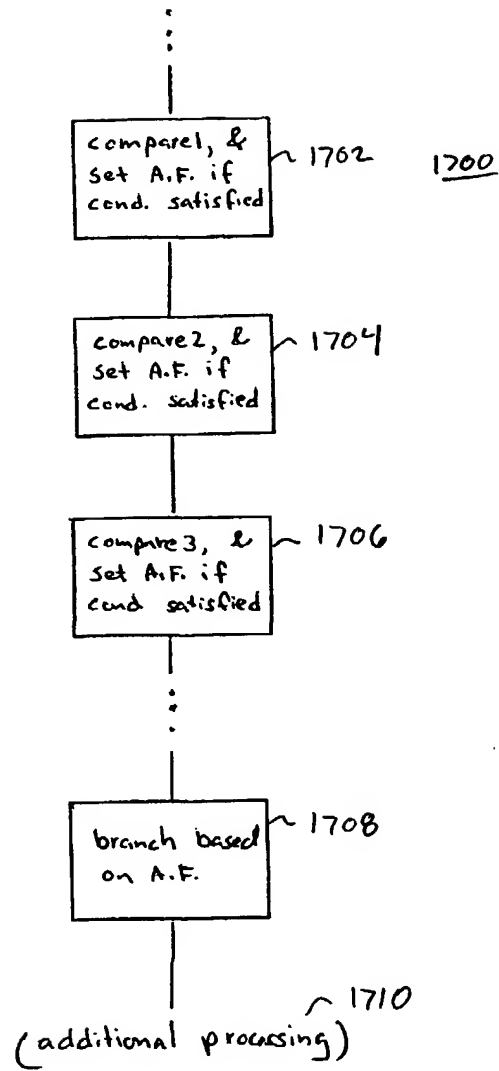


FIG. 90